Test Generation for Ultra-High-Speed Asynchronous Pipelines*

Feng Shi & Yiorgos Makris
Electrical Engineering Dept.
Yale University
New Haven, CT 06520, USA

Steven M. Nowick
Computer Science Dept.
Columbia University
New York, NY 10027, USA

Montek Singh
Computer Science Dept.
University of North Carolina
Chapel Hill, NC 27599, USA

Abstract

We propose a methodology for testing ultra-high-speed asynchronous pipelines, the latest and most promising asynchronous circuit design style. Unlike traditional delay-insensitive asynchronous micro-pipelines, which use slow capture-pass latches, these circuits employ aggressive handshaking protocols and transparent latches between fine-grain pipeline stages, in order to achieve high performance. Their functional robustness, however, relies on certain timing constraints that need to be satisfied. As a result, these circuits are no longer delay-insensitive, which means that stuck-at faults will not always lead to pipeline stalling. In addition, delay faults may result in violation of these timing constraints, thus affecting not only performance, as in delay-insensitive micro-pipelines, but also functional correctness. To address these new challenges, we develop a test method for both stuck-at and timing constraint violation faults in fine-grain ultra-high-speed asynchronous pipelines. The efficiency of the proposed method is demonstrated on MOUSETRAP, a recently developed pipeline for high-speed applications.

1 Introduction

Ever since Ivan Sutherland revitalized interest in asynchronous circuits by introducing *micro-pipelines* [1], numerous alternative styles have been proposed for designing asynchronous pipelines. These efforts are motivated by a number of advantages that asynchronous circuits promise over their synchronous counterparts. Despite their ability to achieve multi-gigahertz clock frequencies, synchronous designs are challenged by problems with clock skew, clock power management, and interfacing across clock domains, which are summarily eliminated in asynchronous designs. In addition, asynchronous pipelines are inherently more robust and can naturally tolerate process variations and fluctuations in operational conditions [2]. Moreover, they exhibit excellent modularity and can interface directly with a multitude of environments through simple handshaking protocols, facilitating reusability. Migration of asynchronous pipelines from research labs to industrial production, however, calls for CAD support in several areas, including test.

A number of methods have been proposed in the past for testing asynchronous pipelines [3, 4, 5, 6, 7, 8]. Most of these methods focus mainly on testing the classic micro-pipeline [1], which is a *delay-insensitive* architecture, i.e. it operates correctly under arbitrary gate and wire delays, except that the standard bundled data timing convention still needs to be observed when datapaths are included. Since then, however, a new generation of ultra-high-speed asynchronous pipelines have been developed, introducing a new set of test challenges which make existing asynchronous test methodologies obsolete. Indeed, these circuits exhibit new faulty behaviors and impose new cost and performance limitations that are not considered in existing test methods.

More specifically, ultra-high-speed asynchronous pipelines typically sacrifice the robustness of a delay-insensitive design style and employ aggressive handshaking protocols in order to achieve high performance. This high performance, however, is obtained at the cost of certain timing constraints that need to be observed for the circuit to function correctly. As a result, traditional testing becomes a more complicated task because not all control stuck-at faults result in pipeline stalling, as in the micro-pipeline architecture [3, 9, 10]. Moreover, delay faults that result in violation of the aforementioned timing constraints also need to be considered. Unlike in the delay-insensitive micro-pipeline, where such faults only result in performance degradation, functional correctness is jeopardized if timing constraints are violated in ultra-high-speed asynchronous pipelines. In addition, these circuits achieve high performance partially due to their very fine-grain pipeline stages, which in the extreme case may contain a single level of gates. Such fine-grain architectures make scan-based test solutions unacceptably expensive and usually unnecessary. In order to develop cost effective test strategies for these circuits, the capabilities of non-scan based methods need to be leveraged before intrusive hardware is added for test.

In this paper, we propose a test method for ultra-high-speed asynchronous pipelines. Our method builds upon a previously developed tool-suite for the class of speed-independent circuits [11, 12, 13], which we extend in order to handle delay-insensitive and quasi-delay-insensitive circuits and to incorporate timing constraints. In addition, we introduce a fault model and a simple design-for-test (DFT) method to support testing for timing constraint violations, which are critical to the correct operation of ultra-high-speed asynchronous pipelines. Moreover, the proposed method guarantees that the fault effects are observed

*This work is partially supported by the DARPA CLASS program.

0-7803-9039-3/$20.00 © 2005 IEEE

in stable states and requires that the test patterns are applied after the circuit stabilizes, in order to facilitate testing using slow testers. The MOUSETRAP [2] pipeline is used for demonstrating the proposed methods.

The rest of this paper is organized as follows. In section 2, we briefly review previously proposed test methods for asynchronous pipelines. In section 3, we introduce the basics of asynchronous pipelines, with emphasis on the details of MOUSETRAP. In section 4, we discuss our test method for stuck-at faults in ultra-high-speed asynchronous pipelines. In section 5, we describe the proposed DFT method which facilitates testing of timing constraint violation faults.

2 Previous Work

Several researchers have studied the problem of testing asynchronous pipelines in the past. Pagey *et al.* [3] proposed a test generation method for stuck-at faults in traditional micro-pipelines. However, timing faults resulting in bundling constraint violation are not considered. Khoche *et al.* [4] and Petlin *et al.* [5] developed full-scan approaches for testing micro-pipelines. Their methods are able to test not only for stuck-at faults but also for delay faults resulting in bundling constraint violation. However, full-scan methods increase the test application time and incur hardware overhead which may be prohibitive for fine-grain asynchronous pipelines. King *et al.* [6] developed a method which generates test sequences for faults inside the C-elements of the micro-pipeline control stages and applies them through the circuit. Roncken *et al.* [7, 8] proposed a partial scan method to test both datapaths and handshake components in asynchronous pipelines. Test patterns are generated not only for gate I/O stuck-at faults, but also for bridging faults. In addition to voltage testing, I_{DDQ} testing is also performed to improve test quality. Furthermore, a handshaking component, $HOLD$, is introduced to hold the circuit in between a handshaking request and its acknowledgement, in order to regain adequate test control. As mentioned earlier, these methods were developed for traditional micro-pipelines and do not address the additional test needs of fine-grain ultra-high-speed asynchronous pipelines.

3 Styles of Asynchronous Pipelines

Before describing the styles of asynchronous pipelines, first we briefly introduce the most common classes of asynchronous circuits, based on their timing assumptions. *Delay-Insensitive* circuits are the most robust, yet the class of such circuits built out of simple gates is rather limited. *Quasi-Delay-Insensitive* circuits are delay-insensitive except that "isochronic forks" are required to build practical circuits using simple gates and operators. An isochronic fork is a forked wire where all branches have exactly the same delay. *Speed-Independent* circuits tolerate arbitrary gate delays, but assume negligible wire delays. *Timed* circuits operate correctly under specific internal and/or environmental timing assumptions such as bounded delays.

Sutherland [1] proposed the classic asynchronous pipeline architecture called *micro-pipeline*. The control circuit for a micro-pipeline, which is shown in Figure 4 (a), is a string of Muller C-elements. Micro-pipeline stages handshake through transition signaling and are *delay-insensitive*. However, this style uses complex and slow capture-pass latches which hinder the performance. To alleviate this problem, a four-phase micro-pipeline control approach was proposed [14] in order to improve performance by employing faster single-phase transparent latches. From then onwards, many variants of micro-pipelines have been developed using alternative control and latch structures [2, 15, 16, 17, 18, 19]. In order to improve performance, most of these asynchronous pipelines employ fine-grained stages and aggressive timing, i.e. they give up the robustness of delay-insensitivity and require that certain timing constraints are met in order to operate correctly.

Among these architectures, we focus on MOUSETRAP [2] (or **M**inimal-**O**verhead **U**ltra-high-**S**p**E**ed **TR**ansition-signaling **A**synchronous **P**ipeline), which we will use as a vehicle to demonstrate the proposed test methodology. The structure of a MOUSETRAP asynchronous pipeline including both control and processing logic is shown in Figure 1. A MOUSETRAP pipeline works under a hybrid protocol [2] – *transition signaling* for the handshaking signals, and *level signaling* for the latch enable signal. In its initial state, the pipeline is empty with all the *done*, *req*, and *ack* signals at a low level, and all its latches in transparent mode. The pipeline communicates with the environment through transition signaling, that is, each transition, whether up or down, is treated as a distinct event. When the first request, or a rise transition on req_{N-1} flow through the pipeline stage $N-1$, $done_{N-1}$ changes to one and En_{N-1} to zero, which closes the latches in stage $N-1$ in order to lock the stage and block any new request. The same process is applied to the data, which is "bundled", i.e. it arrives before the request. Then req_N rises after a certain delay which matches the delay of the processing logic in this stage, and both $done_N$ and ack_N change to one. As a result, En_N changes to zero to close the latches and lock the results, and En_{N-1} changes to one which opens the latches in stage $N-1$ to accept the new request. After a successive request, or a fall transition, flows through stage $N-1$, $done_{N-1}$ changes to zero and En_{N-1} changes to zero to close the latches, and the same process repeats. Note that for each data item there are two transitions on each En signal, one to capture it and one to release it. Therefore, the XNOR gate serves as a *phase converter* which converts the $done$ and ack signals of transition signaling into level control for the transparent latches.

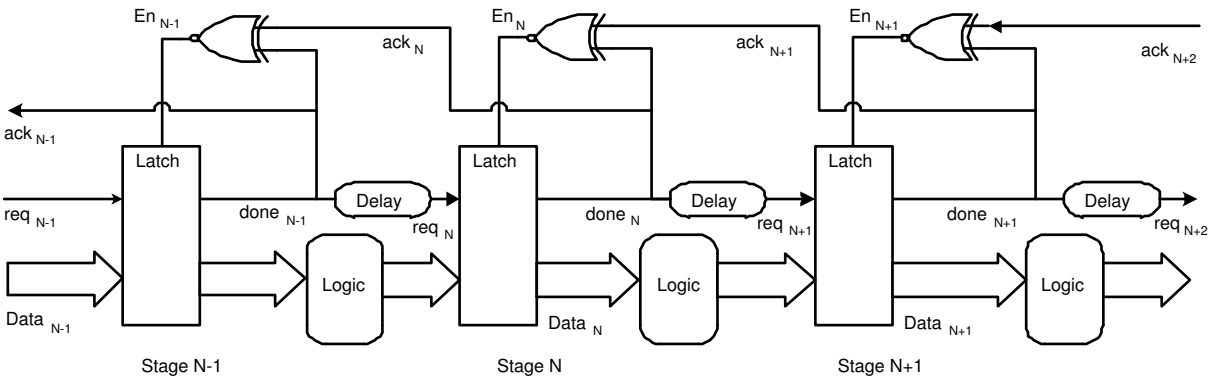


Figure 1. MOUSETRAP Pipeline with Processing Logic

We point out that MOUSETRAP is a fine-grain pipeline in the sense that the depth of the data processing logic can be very shallow, comprising in the extreme case just a single level of gates. MOUSETRAP achieves high performance by using simple and fast transparent latches, while avoiding the extra switching activity of a 4-phase communication [14] to simplify the control circuitry. This aggressive handshaking protocol generates an earlier completion signal but its robustness relies on a few simple one-sided timing constraints.

We should note that numerous even more aggressive structures of fine-grain asynchronous pipelines have also been proposed [2, 20, 21, 22, 23], wherein latches are eliminated altogether. These designs usually employ dynamic logic for the datapaths and exploit the inherent latching properties of dynamic gates. Therefore, by avoiding explicit pipeline latches, the performance is further improved, area is saved, and power consumption is reduced. However, dynamic styles are less noise immune than static styles and most commercial CAD design flows are not geared towards dynamic logic. In contrast, the bundled static styles, such as MOUSETRAP, are able to reuse existing synchronous datapath blocks. Moreover, in some designs [23], delay-insensitive codes such as dual-rail data code are adopted to indicate the data state, hence no dedicated request signal is needed. This style is particularly suitable for *single-gate-level pipelines*, which achieve extremely high throughput by partitioning the datapath in the finest-possible stages with no explicit latches. However, dual-rail styles impose a significant increase in both area and power.

4 Testing Stuck-At Faults

Since stuck-at faults constitute the basic model in both synchronous and asynchronous circuits, we first discuss stuck-at fault testing in asynchronous pipelines, emphasizing on fine-grain ultra-high-speed architectures and in particular on MOUSETRAP. We divide the circuit in two parts, the handshaking logic (control) and the processing logic (datapath), which are treated individually. In order to test the handshaking logic, we extend a tool-suite that we recently developed for testing speed-independent circuits [11, 12, 13]. These extensions include the ability to handle other classes of asynchronous circuits, such as delay-insensitive and quasi-delay-insensitive, and the ability to take timing constraints into consideration. Testing the processing logic is relatively simple based on the transparent latches that are used in MOUSETRAP and only requires conventional combinational ATPG. Results are reported for the traditional micro-pipeline and for both the linear and non-linear MOUSETRAP pipeline architectures.

4.1 Testing the Handshaking Logic

It has been observed in the past [3, 4] that a stuck-at fault on a request or an acknowledge line of a micro-pipeline will prevent the generation of any further events on that line. Hence, the micro-pipeline is halted from progressing any further and the fault can be easily detected. While this is true for delay-insensitive micro-pipelines, ultra-high-speed asynchronous pipelines employ timing assumptions to improve performance and reduce hardware cost and are not delay-insensitive any longer. For instance, the correct operation of MOUSETRAP is based on a couple of timing constraints. Therefore, not all stuck-at faults in the control circuit of MOUSETRAP will halt the pipeline. In addition, many other styles of asynchronous pipelines have been proposed since the traditional micro-pipeline. These styles employ different handshaking protocols and control circuit structures, and operate under different timing constraints. Therefore, a test generation method which is able to handle these new pipeline styles is necessary.

The test generation method that we propose for asynchronous pipelines is based on a tool-suite that we recently developed for speed-independent circuits. The tool-suite performs simulation (SPIN-SIM [11]), ATPG (SPIN-TEST [12]), and test compaction (SPIN-PAC [13]). The heart of this tool-suite is SPIN-SIM, a logic and fault simulator that efficiently detects hazards by extending Eichelberger's classical method [24] to overcome its limitations. In order to

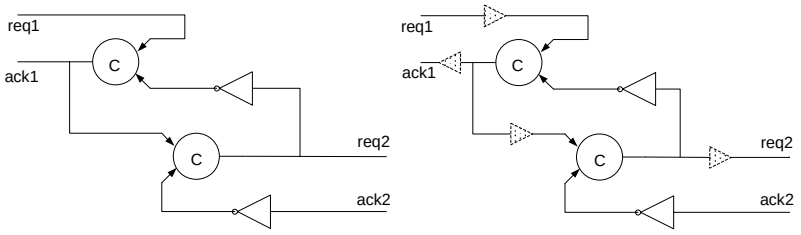


Figure 2. An Delay-Insensitive Circuit (left) and its Equivalent Speed-Independent Circuit (right)

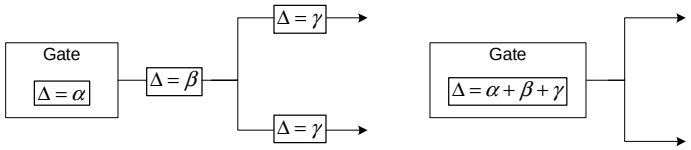


Figure 3. An Isochronic Fork (left) and its Equivalent Speed-Independent Circuit (right)

improve simulation accuracy, SPIN-SIM adopts a 13-valued algebra [25, 26], maintains the relative order of causal signal transitions, and unfolds time frames judiciously. In addition, complex gates are handled through replacement by pseudo-gate equivalents with regards to functionality, timing and faulty behavior. SPIN-TEST uses SPIN-SIM and an A* search algorithm in order to perform fault-simulation-based ATPG. Finally, SPIN-PAC employs several heuristics to combine multiple single-input-change (SIC) vectors into a single multiple-input-change (MIC) vector, as well as an Integer-Linear-Programming (ILP) formulation to compact independent test sequences via pruning or elimination.

Although this tool-suite was developed for speed-independent circuits, it can be extended to handle other classes of asynchronous circuits. First, we employ the method in [27] to simulate delay-insensitive circuits using SPIN-SIM. Delay-insensitive circuits operate correctly under the unbounded delay model, which allows arbitrary gate and wire delays. Unlike in speed-independent circuits, the wires in delay-insensitive circuits also have unbounded delay. Therefore, we preprocess and transform the circuit into a new circuit by inserting buffers on the wires. As a result, under the speed-independent timing model, the transformed circuit exhibits the same timing properties as the original circuit. Therefore, the original circuit can be handled by simulating the transformed circuit in SPIN-SIM. In Figure 2, we give an example of a delay-insensitive circuit (on the left) and the transformed circuit used for simulation by SPIN-SIM (on the right). The inserted buffers, which are used to mimic the timing of the original delay-insensitive circuit, are shown in dashed lines. We also note that it is not necessary to insert a buffer in every segment of a wire, since some of them can be combined with the delay of the gates that drive them.

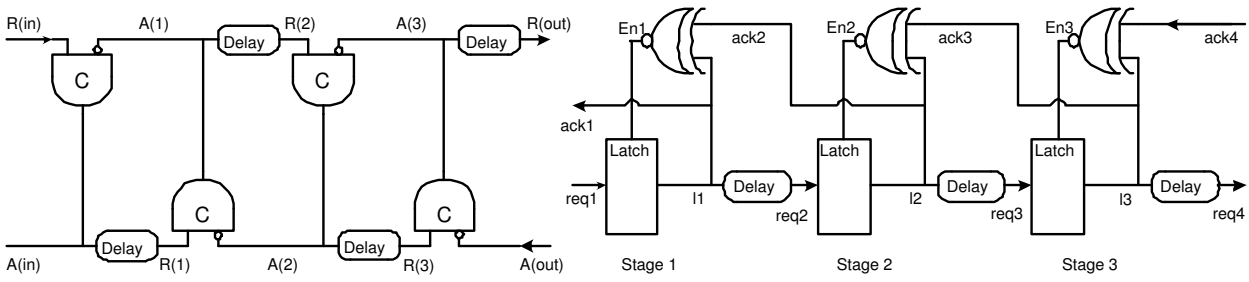
Quasi-delay-insensitive circuits can also be simulated by SPIN-SIM by preprocessing them in a similar way. However, such circuits employ *isochronic forks*, in which the delay on all fanout branches of a wire is assumed to be equal. These isochronic forks need to be handled differently. In general, we transform an isochronic fork into an equivalent speed-independent circuit form, as shown in Figure 3. Therefore, it is not necessary to insert buffers in these isochronic forks. Of course, buffers are still inserted in other parts of the circuit, including non-isochronic forks.

We also extended SPIN-SIM to simulate asynchronous pipelines with particular timing constraints. SPIN-SIM has an inherent feature that comes in handy for this purpose. More specifically, it uses time-stamps to keep track of the relative signal order while simulating speed-independent circuits. This relative signal order is taken into account when evaluating a gate during simulation and increases dramatically the simulation accuracy by eliminating false hazard identification. Based on this capability, timing constraints can be easily incorporated as additional relative orderings of signals.

The transformation process and the extensions described above have been automated, allowing seamless application of SPIN-SIM to speed-independent, delay-insensitive and quasi-delay insensitive circuits, while also taking into account all relevant timing constraints, such as isochronic forks, using time stamps [11]. In this way, logic and fault simulation of aggressive asynchronous pipelines via SPIN-SIM is enabled. Consequently, the SPIN-TEST simulation-based ATPG and the SPIN-PAC test compaction method are also readily available for these circuits.

4.2 Testing the Processing Logic

When a MOUSETRAP pipeline is empty, all request and acknowledge signals are at a low level and all the latches are transparent. Therefore, all the processing logic blocks are interconnected in a cascade and can be treated as a single logic block for the purpose of testing. Suppose that C



(a) Control Circuit for a Micropipeline

(b) Control Circuit for a MOUSETRAP

Figure 4. Control Circuits for Two Asynchronous Pipelines

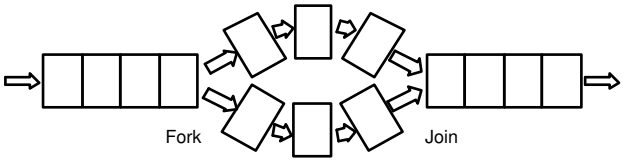


Figure 5. A Non-linear MOUSETRAP

represents the whole block of logic obtained by interconnecting all logic blocks in the pipeline stages in a cascade. Then, any combinational ATPG tool can be used to generate test patterns for stuck-at faults in C. The test patterns are subsequently applied to the input of the pipeline when the pipeline is empty and without sending any request signals. The responses are observed at the output of the pipeline and compared to the expected fault-free responses.

Pagey etc. [3] proposed a similar method for testing stuck-at faults in the processing logic of micro-pipelines, wherein all latches are also in the pass mode when the micro-pipeline is empty. Yet others [4] argued that the time required to generate test vectors for the whole lumped logic block may be very large and the test pattern generator may fail to generate tests for some otherwise detectable faults because of sheer circuit size. Therefore, they proposed a test method based on a full scan path. However, a full scan method may not be appropriate for ultra-high-speed pipelines such as MOUSETRAP, since these pipelines are very fine-grain. Each stage may only contain one level of gates in the extreme case. Hence, fully scanning all the pipeline latches is not only unnecessary but would also incur considerable hardware overhead. Nevertheless, for very long pipelines, a partial scan technique may be worth exploring for reducing the complexity of the lumped processing logic for ATPG purposes.

4.3 Experimental Results

The extended tool-suite for testing stuck-at faults in asynchronous pipelines is implemented in C. The input netlist is in ISCAS89 format and the stuck-at fault list can be either provided through a file or generated automatically. In the latter case, all single stuck-at faults on gate inputs and outputs are injected and g-equivalent [28] faults are collapsed. An optional random ATPG phase is also

Type of Pipeline	No. of Faults	Collapsed Faults	Fault Coverage	Test Length
Micropipeline	36	28	100%	4
MOUSETRAP	40	34	100%	7
Non-linear	182	154	96.1%	15

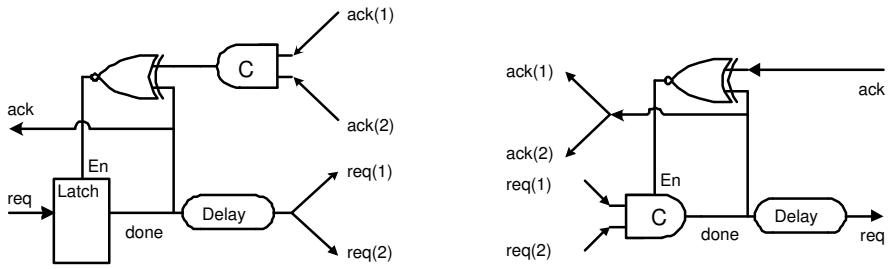
Table 1. Experimental Results for Stuck-at Faults

provided, implementing the algorithm of [29] with a user-defined termination condition (i.e. the maximal number of consecutive random test vectors that detect no faults). Then, the simulation-based, deterministic test pattern generation phase is performed for each remaining fault and the generated test patterns are compacted, preserving fault coverage.

We applied our method on three asynchronous pipelines. The first one is the control part of the traditional micropipeline, shown in Figure 4 (a). The second one is the control part of the MOUSETRAP pipeline, shown in Figure 4 (b). The third one is a non-linear MOUSETRAP pipeline [2], illustrated in Figure 5. The fork and join stages of this non-linear architecture are shown in Figure 6. We note that our method does not target faults inside the C-Elements of the micro-pipeline, since the method proposed in [6] to test for these faults can be applied in parallel with our tool-suite. The experimental results are reported in Table 1. The names of the asynchronous pipelines are listed in the first column, and the number of total stuck-at faults and g-equivalent collapsed faults are shown in the second and third columns, respectively. The achieved fault coverage is listed in the fourth column, and the number of test patterns after test compaction is shown in the fifth column.

The results show that, as expected, testing stuck-at faults in the control circuit of a micro-pipeline is relatively easy. Since a micro-pipeline is delay-insensitive and a stuck-at fault on a C-element allows at most one event to be generated, any stuck-at fault in the control circuit can be detected by applying two request events and observing the outputs. So the test pattern for all stuck-at faults is $00 \rightarrow 10 \rightarrow 00 \rightarrow 01$, where the first bit represents the value on $R(in)$ and the second bit represents the value on $A(out)$.

Testing the control circuit of MOUSETRAP, however, is not as easy, since some stuck-at faults, such as the stuck-at-1 fault on line $En1$ in Figure 4 (b), do not cause the pipeline to halt. Nevertheless, our method still achieves a fault cov-



(a) Control Circuit for a Fork Stage

(b) Control Circuit for a Join Stage

Figure 6. Control Circuits for a Fork Stage and a Join Stage

erage of 100%. The corresponding test sequence generated by our method is $00 \rightarrow 10 \rightarrow 00 \rightarrow 10 \rightarrow 00$, where the first bit represents the value on $req1$ and the second bit represents the value on $ack4$. In order to activate the fault by closing the latch in the first stage, the sequence first sends three requests on the left interface without any acknowledgement on the right interface. Then, an additional request is applied to test whether the latch is closed or not. In the fault-free circuit the latch is closed, hence $ack1$ remains at logic one, while in the faulty circuit the latch is still open, and $ack1$ changes to zero. Unfortunately, the length of the test sequence for this type of fault is $N + 2$, where N is the number of pipeline stages. Therefore, for a very long MOUSETRAP, DFT strategies, such as the one described in section 5, may be necessary to reduce test length.

In the non-linear pipeline, this type of fault in the stages of the fork branches is undetectable and, therefore, fault coverage is less than 100%. In order to detect the fault, consecutive requests need to be applied without any acknowledgement. In the linear case, the fault-free pipeline will be filled while the faulty pipeline will not. In the non-linear case, however, both the fault-free and faulty pipelines will be filled. This happens because, despite the inability to fill the faulty branch, the non-faulty branch of the fork will still be filled. And if either of the two branches is filled, the rest of the pipeline will also be filled. Therefore, DFT strategies may be needed in complex pipeline structures to improve testability.

5 Testing Timing Constraint Violation Faults

In order to achieve ultra-high performance, aggressive asynchronous pipelines such as MOUSETRAP often assume timing constraints to simplify the handshaking protocol and reduce control hardware. For example, in MOUSETRAP, two timing constraints must be satisfied for correct operation of the pipeline: *data overrun* and *setup time*. Even if these timing constraints are satisfied and verified during design, they may be violated due to delay faults caused by manufacturing defects. Therefore, delay faults may cause aggressive asynchronous pipelines to fail, while in delay-insensitive styles they may only degrade their performance. In this section, we discuss how to test whether these timing constraints are met.

5.1 Timing Constraint Violation Fault Model

Timing constraints in asynchronous circuits are typically expressed in the form of $t_1 > t_2$, where t_1, t_2 represent the delay along a certain path, a certain predefined period of time, or their combination. For example, the timing constraint for data overrun in MOUSETRAP is:

$$t_{XNOR_{N-1}\uparrow} + t_{Lt_{N-1}} + t_{logic_{N-1}} > t_{XNOR_N\downarrow} + t_{hold} \quad (1)$$

In other words, since ack_N and $done_N$ are generated in parallel, the path from ack_N to the data inputs of stage N must be longer than the time it takes to close the latch of stage N, plus a hold time t_{hold}. Otherwise, the data in stage N may be overwritten by new data. Under the *timing constraint violation fault model*, a circuit is faulty if its actual performance violates any of the given timing constraints. For instance, if a fault violates timing assumption (1), then in the faulty circuit the opposite inequality will hold, i.e. new data will be allowed to arrive to stage N before its latch is closed, thus overwriting the current data of stage N. The fault that violates the *setup time* constraint in MOUSETRAP can also be modelled in a similar way. The number of timing constraint violation faults in a circuit equals the number of its one-sided timing constraints.

5.2 Difficulties in Testing Timing Constraints

Timing constraint violation faults can be very challenging not only for test generation but also for test application. The two key difficulties are outlined below.

First, high-speed automatic test equipment (ATE) is very expensive, if at all available, for the operational speed targeted by modern ultra-high-speed asynchronous pipelines. For example, in order to test the data overrun timing constraint in MOUSETRAP, consecutive test patterns are needed. New data in the input of stage $N - 1$ is required to be available when the acknowledge signal arrives from stage N, so that it will overwrite the previous data in stage

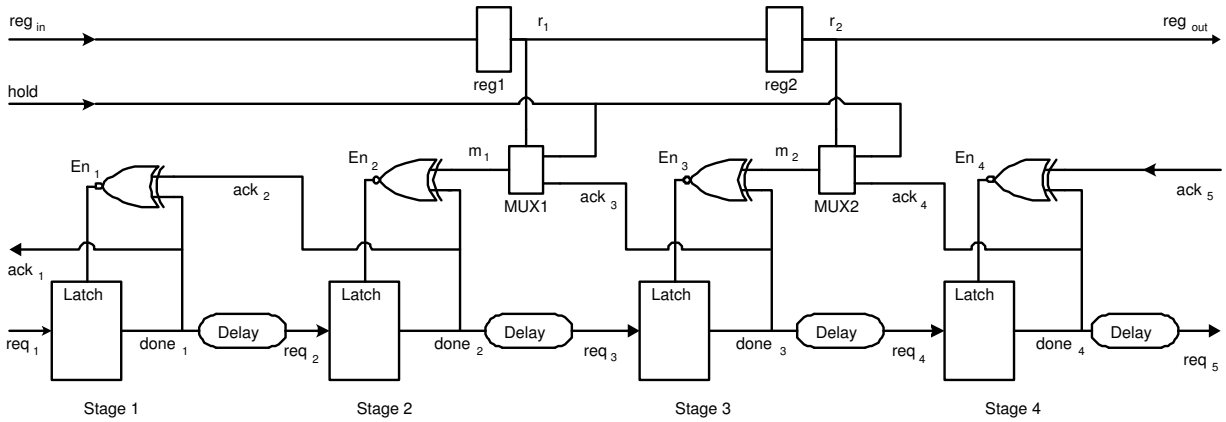


Figure 7. Testing Data Overrun in Asynchronous Pipelines

N in the presence of a timing constraint violation fault. Since the latency between two pipeline stages can be in the order of just a few gates, it is unreasonable to expect that ATE will be able to apply the consecutive test patterns so quickly. The same problem occurs in delay testing of high speed synchronous circuits, since ATE in the test floor is usually several times slower than the new synchronous designs that it tests. This problem is often dealt with by moving some of the ATE capabilities on the chip in the form of DFT hardware. Similar strategies may be required for testing high-speed asynchronous pipelines using slow ATE.

Second, unlike stuck-at faults, timing constraint violation faults may cause hazards in the circuit, which may prevent the propagation of fault effects. While testing the data overrun timing constraint, for example, new data comes in before the old data settles. Therefore, what exactly will be captured in the latches of the next stage is unknown. As a result, since the fault response is non-deterministic, the fault can, at best, be *potentially detected*. Thus, smart ways for ensuring deterministic fault responses are also required, in order to detect timing constraint violation faults.

5.3 Proposed Method

In order to test the data overrun timing constraint in MOUSETRAP, we propose the DFT strategy shown in Figure 7[1]. The main source of difficulty in testing timing constraint violation faults in MOUSETRAP is the weak controllability of the pipeline through the primary inputs. Once a new request arrives, a series of operations occurs through the whole pipeline, with no way to pause in between. As a result, these unbridled operations often yield an unknown response due to hazards injected by a timing constraint violation fault. To improve controllability, we insert a two-input multiplexer in each pipeline stage except the first and the last. As illustrated in Figure 7, the multiplexer is inserted in the acknowledge signal coming from the next pipeline stage. In normal operation mode, the select input of the multiplexer is set to zero so that the acknowledge signal from the next stage will pass through, as in the original MOUSETRAP. In test mode, however, the select input of the multiplexer is set to one, so that its output can be controlled by the user through the primary input $hold$. Thus, the user can control the XNOR gate and the latch and, hence, can pause or resume the pipeline at will.

Data overrun faults in MOUSETRAP can now be tested in the following way. Assume that there is a timing constraint violation fault in the second stage of the circuit shown in Figure 7, and that the circuit is in its initial state. In order to test for this fault, the select input to MUX1 is set to one through the shift registers and the primary input $hold$ is set to one. Thus $m_1 = 1$, $done_2 = 0$, and $En_2 = 0$ and the latch of the second stage is closed. Then, a one is applied to input req_1, followed by a zero on req_1 after the circuit stabilizes, so that there are two requests blocked by the closed latch of the second stage. Then, $hold$ is set to zero, and En_2 changes to one in order to accept a new request. The fault-free and faulty circuits have different responses as follows.

If there is no data overrun fault in the second stage, only one request propagates through the second latch; this happens because after the first request goes through the latch, En_2 changes to zero and blocks the second request. Even after the first request has passed through the third stage, the latch in the second stage will not open again because the acknowledge signal from the third stage is blocked by the multiplexer. Therefore, after the first request propagates through the complete pipeline to the last stage, the primary output req_5 will be one even after the primary input ack_5 changes to one. A waveform of these signals during this process is illustrated in Figure 8 (a).

However, as shown in the waveform of Figure 8 (b), incorrect responses are observed if a data overrun fault exists in the second stage of the pipeline. In this case, after the first request goes through the second stage, the latch of the

[1]The processing logic is omitted in Figure 7

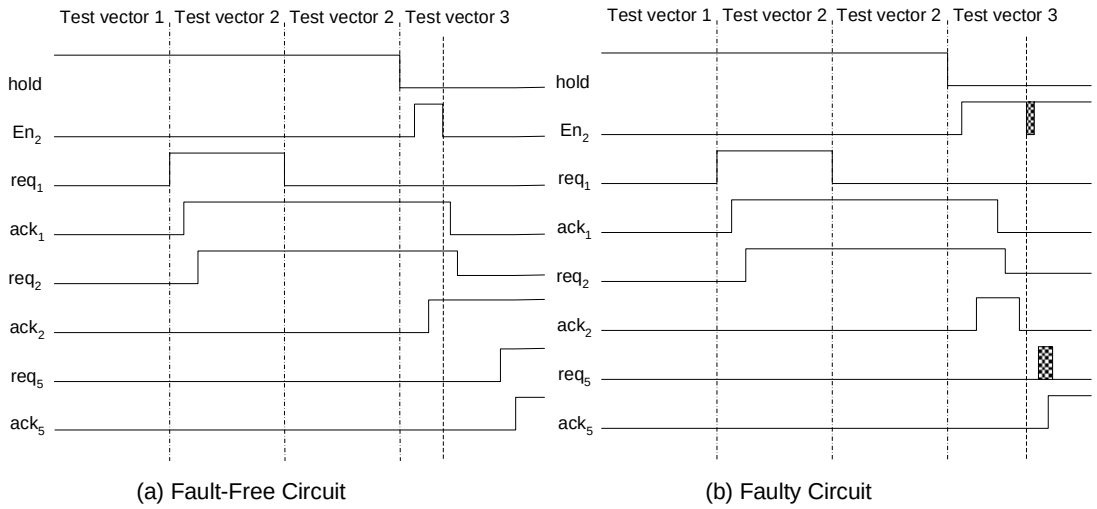


Figure 8. Waveforms During Testing a Timing Constraint

first stage is opened and the second request arrives before the latch in the second stage is closed. Therefore, the second request also goes through the second stage and the fault is activated. In this case, the primary output req_5 is finally zero, which is different from the correct response. As a result, this timing constraint violation fault is detected. Notice that sending two consecutive requests from the ATE may not activate the fault, since the ATE may not be fast enough in applying the second request after the acknowledge signal arrives. Hence, the second request may arrive after the second stage latch is closed and the data overrun fault will not be activated.

Now let us prove that, in the faulty case, the primary output req_5 is guaranteed to be zero. If the data overrun fault exists in the second stage, both requests will pass through the latch of the second stage. There are three possible cases for the propagation of the two requests. In the first case, the two requests are captured successively by the next stage, as in normal operation. Then, they propagate, as usual, to the last pipeline stage sequentially, and after ack_5 acknowledges the first request on req_5, the final response on req_5 is zero. In the second case, the two requests cancel each other and disappear from the line. In this case, the primary output req_5 remains at zero. In the third case, if neither of the previous cases happens and the two requests pass through all the stages together, then req_5 is still zero in the end. In conclusion, the primary output req_5 is guranteed to be zero in the presence of the data overrun fault.

Data overrun faults in other pipeline stages can be tested in a similar way. Since there are $N-2$ multiplexers inserted in an N-stage pipeline, there is a total of $N-2$ select inputs to these multiplexers. To control them through a single primary input pin, we introduce a shift register to which we connect them, as shown in Figure 7. The shift register may operate either synchronously or asynchronously in test mode, so we ommit showing its control signals in Figure 7. By shifting a one through the register, we can test successively the data overrun faults of all pipeline stages.

Table 2 lists the test patterns required for testing three single data overrun timing constraint violation faults in the MOUSETRAP pipeline of Figure 7. The first 5 patterns are used for testing the data overrun fault in the second stage, the next 5 for the one in the third stage, and the last 4 for the one in the last stage. Signal reg_{shift} is the control signal of the shift register; a high level on this signal causes a one-bit shift. Note that race conditions may exist in the circuit when the 6th pattern is applied, however, they do not influence the response. The data overrun fault of the first stage is ignored here, mainly because new requests and data are applied from the environment, which usually has higher delay than the internal path.

The proposed DFT method incurs overhead both in hardware and in performance. The additional multiplexers and shift registers contribute to the hardware overhead in, roughly, linear proportion to the number of pipeline stages. In terms of performance, the proposed method has no influence on *forward latency*, but increases *cycle time*. Forward latency is the time it takes a data item to pass through an initially empty pipeline. Thus, the pipeline latency per stage, L, is the sum of the latch delay and the logic delay, as in the original case:

$$L = t_{Lt} + t_{logic} \tag{2}$$

Cycle time is the interval between successive items emerging from the pipeline when it is operating at maximum speed. Since a multiplexer is inserted in the path from the acknowledge signal to the $XNOR$ gate in each stage, it increases the cycle time of a stage N, which is now:

$$T = 2 \times t_{Lt} + t_{logic} + t_{MUX} + t_{XNOR\uparrow} \tag{3}$$

In order to reduce this performance overhead, we are currently looking into ways to optimize the proposed DFT by merging the inserted MUXs with the attached XNOR gates.

No.	Test Pattern					Good Response		Faulty Response		Comments
	req_{in}	req_{shift}	$hold$	req_1	ack_5	ack_1	req_5	ack_1	req_5	
1	1	1	1	0	0	0	0	0	0	Close 2nd latch
2	0	0	1	1	0	1	0	1	0	Apply 1st request
3	0	0	1	0	0	1	0	1	0	Apply 2nd request
4	0	0	0	0	0	0	1	0	X	Pass 1st request through 2nd latch
5	0	0	0	0	1	0	1	0	0	Observe the response
6	0	1	1	0	1	0	0	0	0	Pass 2nd request, then close 3rd latch
7	0	0	1	1	0	1	0	1	0	Apply 3rd request
8	0	0	1	0	0	0	0	0	0	Apply 4th request
9	0	0	0	0	0	0	1	0	X	Pass 3rd request through 3rd latch
10	0	0	0	0	1	0	1	0	0	Observe the response
11	0	1	X	0	1	0	0	0	0	Pass 4th request, then close 4th latch
12	0	0	X	1	1	1	0	1	0	Apply 5th request
13	0	0	X	0	1	0	0	0	0	Apply 6th request
14	0	0	X	0	0	0	1	0	0	Pass 5th request, observe the response

Table 2. Test Patterns for Timing Constraint Violation Faults in MOUSETRAP

Other timing constraints must also be satisfied for correct operation of MOUSETRAP. The *setup time* constraint requires that once a latch is enabled and receives new data at its inputs, it must remain transparent long enough for the data to pass through. This means that the path from req_N to deasserting En_N must be longer than the setup time, t_{su}:

$$t_{req_N \rightarrow done_N} + t_{XNOR_N \downarrow} > t_{su} \quad (4)$$

Also, the standard asynchronous *bundled data* scheme requires that req_N must arrive at stage N after the data inputs of this stage have stabilized. Therefore, the latency of the delay element must match the worst-case delay through the combinational block. Testing these constraints is relatively simple, since any violation will lead the latches to lock erroneous data, which will propagate to the primary outputs.

Suppose that we are testing the setup time constraint for the latch in the second stage, which drives data line a as shown in Figure 9. First we test if the setup time constraint holds for a rising transition. Consider the combinational logic C obtained by removing the pipeline latches and cascading the processing logic blocks. Test generation proceeds in two phases. In the first phase, we find a test pattern v_1 for the combinational logic C that sets line a to zero. Then, in the second phase, we find a test pattern v_2 for C that detects a stuck-at zero fault on line a. During test application, v_1 is first applied to the empty pipeline without any request signals. Since the pipeline is empty, all latches are transparent and thus, after applying v_1, line a is set to zero. Then, the latches in the second stage are closed through the inserted multiplexer. After that, test pattern v_2 is applied with a request signal. Finally, the latches in the second stage are opened. Since v_2 detects a stuck-at zero fault on line a, it causes a rising transition on line a. Therefore, if the setup time constraint is violated, the latch driving a will latch a zero instead of a one, activating the fault. Again, since v_2 is able to detect a stuck-at zero fault on line a in C, the fault effect propagates all the way to the primary outputs, and thus, the timing constraint violation fault is detected. Notice that we close the latches in the second stage before applying v_2. Otherwise, the data may go through the latches of the second stage and set line a to one before the request arrives to the second stage, in which case the timing constraint violation fault will not be activated. If the circuit has asymmetric rising and falling delays, we can test the setup time constraint for a falling transition in a similar way. Also, since either a rising or a falling transition may represent a request event, the test should be performed under both types.

Lastly, we illustrate how to test the timing constraint in the basic asynchronous bundled data scheme. Suppose there is a *segment delay fault* in the third stage of the pipeline, as shown in Figure 9, and this fault causes the delay of the processing logic of the third stage to exceed the delay of the request signal. In order to generate test patterns for this fault, we target the segment delay fault in the combinational logic C obtained by removing the pipeline latches and cascading the processing blocks of each stage. The obtained test patterns, say v_1 and v_2, are the main patterns needed for testing this timing constraint. Test application is similar as in the previous case of testing the setup time constraint. First, v_1 is applied to an empty pipeline without any request signals. Then, the latches of the third stage are closed. After that, v_2 is applied, and then the latches in the third stage are opened. At this time, the segment delay fault is activated and if the delay in processing logic exceeds the delay of the request signal, erroneous data is latched in the fourth stage. Since the patterns are able to detect the segment delay fault in C, the fault effect propagates to the primary outputs and the timing constraint violation fault is detected.

6 Conclusion

The latest ultra-high-speed asynchronous pipeline architectures impose new test requirements for ensuring their operational health. These requirements are not sufficiently addressed by previously proposed test methodologies, which focus on the traditional micro-pipeline architecture. The test method described in this paper fills this gap by pro-

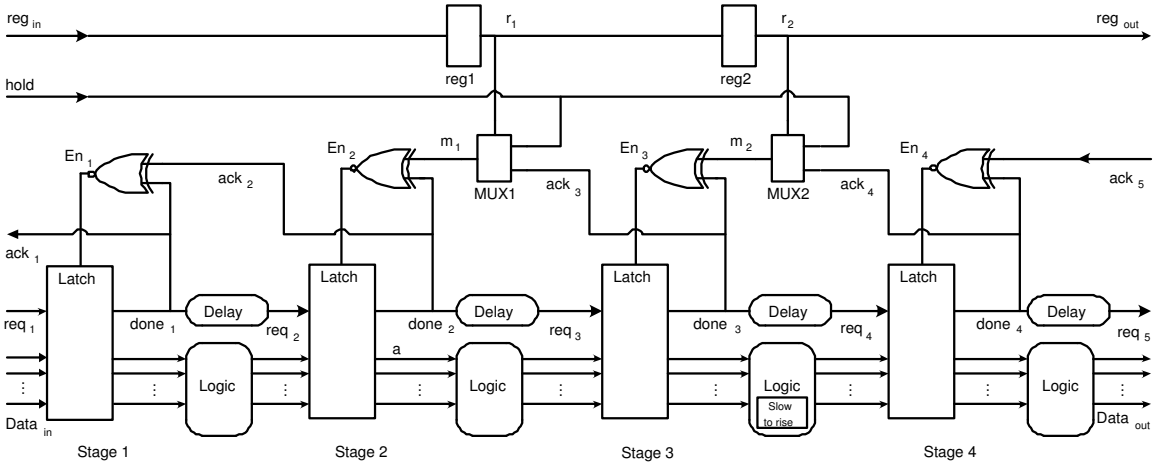


Figure 9. Testing Other Timing Constraints in Asynchronous Pipelines

viding the ability to test for both stuck-at and timing assumption violation faults in modern ultra-high-speed asynchronous pipelines. The proposed ATPG and DFT methods are evaluated on MOUSETRAP, demonstrating their efficiency and pinpointing several directions for improvement.

References

[1] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, June 1989.

[2] M. Singh and S. M. Nowick, "MOUSETRAP: Ultra-high-speed transition-signaling asynchronous pipelines," in *Proc. International Conf. Computer Design (ICCD)*, Nov. 2001, pp. 9–17.

[3] S. Pagey, G. Venkatesh, and S. Sherlekar, "Issues in fault modeling and testing of micropipelines," in *Proc. of the Asian Test Symposium*, Hiroshima, Japan, Nov. 1992.

[4] A. Khoche and E. Brunvand, "Testing micropipelines," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*, Nov. 1994, pp. 239–246.

[5] O. A. Petlin and S. B. Furber, "Scan testing of micropipelines," in *Proc. IEEE VLSI Test Symposium*, May 1995, pp. 296–301.

[6] M. King and K. Saluja, "Testing micropipelined asynchronous circuits," in *Proc. International Test Conference*, Oct. 2004, pp. 329–338.

[7] M. Roncken and E. Bruls, "Test quality of asynchronous circuits: A defect-oriented evaluation," in *Proc. International Test Conference*, Oct. 1996, pp. 205–214.

[8] M. Roncken, E. Aarts, and W. Verhaegh, "Optimal scan for pipelined testing: An asynchronous foundation," in *Proc. International Test Conference*, Oct. 1996, pp. 215–224.

[9] Peter Beerel and Teresa Meng, "Semi-modularity and self-diagnostic asynchronous control circuits," in *Advanced Research in VLSI*, Carlo H. Séquin, Ed. Mar. 1991, pp. 103–117, MIT Press.

[10] Alain J. Martin and Pieter J. Hazewindus, "Testing delay-insensitive circuits," in *Advanced Research in VLSI*, Carlo H. Séquin, Ed. 1991, pp. 118–132, MIT Press.

[11] F. Shi and Y. Makris, "SPIN-SIM: Logic and fault simulation for speed-independent circuits," in *Proc. of International Test Conference*, Oct. 2004, pp. 597–606.

[12] F. Shi and Y. Makris, "SPIN-TEST: Automatic test pattern generation for speed-independent circuits," in *Proc. of International Conference on Computer Aided Design*, Nov. 2004, pp. 903–908.

[13] F. Shi and Y. Makris, "SPIN-PAC: Test compaction for speed-independent circuits," in *Proc. of Asia and South Pacific Design Automation Conference*. Jan. 2005, pp. 71–74, IEEE Computer Society Press.

[14] P. Day and J. Viv Woods, "Investigation into micropipeline latch design styles," *IEEE Transactions on VLSI Systems*, vol. 3, no. 2, pp. 264–272, June 1995.

[15] K. Y. Yun, P. A. Beerel, and J. Arceo, "High-performance asynchronous pipeline circuits," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. Mar. 1996, IEEE Computer Society Press.

[16] C. E. Molnar, I. W. Jones, W. S. Coates, J. K. Lexau, S. M. Fairbanks, and I. E. Sutherland, "Two FIFO ring performance experiments," *Proc. of the IEEE*, vol. 87, no. 2, pp. 297–307, Feb. 1999.

[17] I. Sutherland and S. Fairbanks, "GasP: A minimal FIFO control," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. Mar. 2001, pp. 46–53, IEEE Computer Society Press.

[18] J. Ebergen, "Squaring the FIFO in GasP," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. Mar. 2001, pp. 194–205, IEEE Computer Society Press.

[19] S. Schuster, W. Reohr, P. Cook, D. Heidel, M. Immediato, and K. Jenkins, "Asynchronous interlocked pipelined CMOS circuits operating at 3.3–4.5 ghz," in *Proc. ISSCC*. Feb. 2000, pp. 194–205, IEEE Computer Society Press.

[20] T. E. Williams, *Self-Timed Rings and their Application to Division*, Ph.D. thesis, Stanford University, June 1991.

[21] M. Singh and S. M. Nowick, "Fine-grain pipelined asynchronous adders for high-speed DSP applications," in *Proc. of the IEEE Computer Society Workshop on VLSI*. Apr. 2000, pp. 111–118, IEEE Computer Society Press.

[22] M. Singh and S. M. Nowick, "High-throughput asynchronous pipelines for fine-grain dynamic datapaths," in *Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems*. Apr. 2000, pp. 198–209, IEEE Computer Society Press.

[23] U. Cummings, "Pivotpoint: clockless crossbar switch for high-performance embedded systems," in *Proc. IEEE Micro*. Mar. 2004, pp. 48–59, IEEE Computer Society Press.

[24] E. B. Eichelberger, "Hazard detection in combinational and sequential switching circuits," *IBM Journal of Research and Development*, vol. 9, no. 2, pp. 90–99, 1965.

[25] T. J. Chakraborty, V. D. Agrawal, and M. L. Bushnell, "Delay fault models and test generation for random logic sequential circuits," in *Proc. of the 29th Design Automation Conference*, 1992, pp. 165–172.

[26] D. S. Kung, "Hazard-non-increasing gate-level optimization algorithms," in *International Conference on Computer Aided Design*, 1992, pp. 631–634.

[27] D. L. Dill, *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*, MIT Press, Cambridge, MA, 1989.

[28] J. E. Chen, C. L. Lee, and W. J. Shen, "Single-fault fault-collapsing analysis in sequential logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 12, pp. 1559–1568, 1991.

[29] F. Shi and Y. Makris, "Fault simulation and random test generation for speed-independent circuits," in *Proc. of the 2004 Great Lakes Symposium on VLSI*, Apr. 2004, pp. 127–130.